Import Necessary Libraries

```python
# Import Data Science Libraries
import numpy as np
import pandas as pd
import tensorflow as tf
from sklearn.model_selection import train_test_split
import itertools
import random

# Import visualization libraries
import matplotlib.pyplot as plt
import matplotlib.cm as cm
import cv2
import seaborn as sns

# Tensorflow Libraries
from tensorflow import keras
from tensorflow.keras import layers,models
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization
from tensorflow.keras.callbacks import Callback,
EarlyStopping,ModelCheckpoint, ReduceLROnPlateau
from tensorflow.keras import Model
from tensorflow.keras.layers.experimental import preprocessing
from tensorflow.keras.optimizers import Adam

# System libraries
from pathlib import Path
import os.path

# Metrics
from sklearn.metrics import classification_report, confusion_matrix

sns.set_style('darkgrid')

# Seed Everything to reproduce results for future use cases
def seed_everything(seed=42):
    # Seed value for TensorFlow
    tf.random.set_seed(seed)

    # Seed value for NumPy
    np.random.seed(seed)

    # Seed value for Python's random library
    random.seed(seed)

    # Force TensorFlow to use single thread
    # Multiple threads are a potential source of non-reproducible
results.
    session_conf = tf.compat.v1.ConfigProto(
        intra_op_parallelism_threads=1,
        inter_op_parallelism_threads=1
    )

    # Make sure that TensorFlow uses a deterministic operation
wherever possible
    tf.compat.v1.set_random_seed(seed)

    sess =
tf.compat.v1.Session(graph=tf.compat.v1.get_default_graph(),
config=session_conf)
    tf.compat.v1.keras.backend.set_session(sess)

seed_everything()
```

🤙 Create helper functions

```
!wget https://raw.githubusercontent.com/mrdbourke/tensorflow-deep-
learning/main/extras/helper_functions.py

# Import series of helper functions for our notebook
from helper_functions import create_tensorboard_callback,
plot_loss_curves, unzip_data, compare_historys, walk_through_dir,
pred_and_plot
```

```
--2024-04-15 08:00:10--
https://raw.githubusercontent.com/mrdbourke/tensorflow-deep-learning/
main/extras/helper_functions.py
Resolving raw.githubusercontent.com (raw.githubusercontent.com)...
185.199.110.133, 185.199.111.133, 185.199.109.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|
185.199.110.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 10246 (10K) [text/plain]
Saving to: ‘helper_functions.py’

helper_functions.py 100%[===================>]  10.01K  --.-KB/s    in
0s

2024-04-15 08:00:10 (70.3 MB/s) - ‘helper_functions.py’ saved
[10246/10246]
```

Load and Transform Data

```
BATCH_SIZE = 32
TARGET_SIZE = (224, 224)

# Walk through each directory
dataset = "/kaggle/input/animal-kingdom/raw-img"
walk_through_dir(dataset)
```

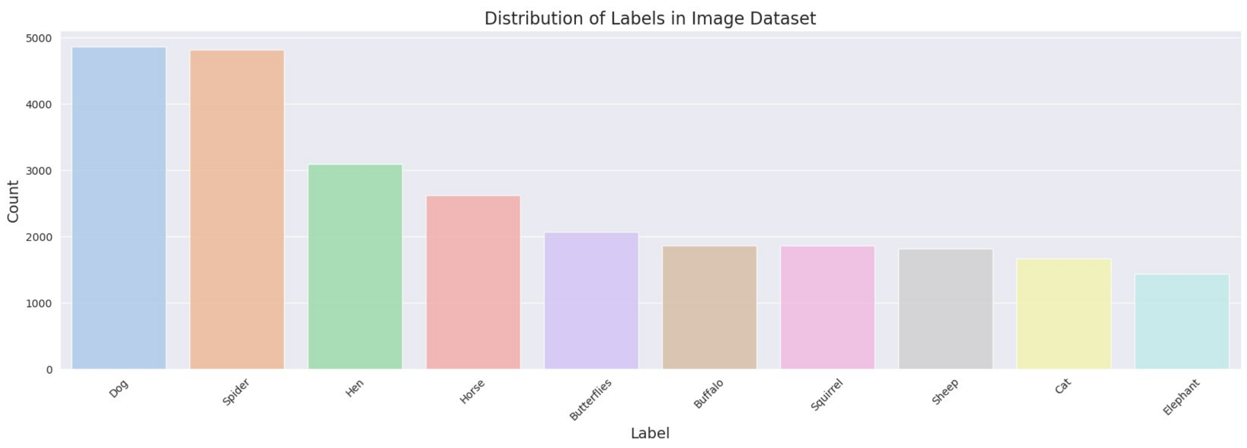
```
There are 10 directories and 0 images in '/kaggle/input/animal-
kingdom/raw-img'.
There are 0 directories and 4821 images in '/kaggle/input/animal-
kingdom/raw-img/Spider'.
There are 0 directories and 2623 images in '/kaggle/input/animal-
kingdom/raw-img/Horse'.
There are 0 directories and 4863 images in '/kaggle/input/animal-
kingdom/raw-img/Dog'.
There are 0 directories and 1866 images in '/kaggle/input/animal-
kingdom/raw-img/Buffalo'.
There are 0 directories and 2112 images in '/kaggle/input/animal-
kingdom/raw-img/Butterflies'.
There are 0 directories and 3098 images in '/kaggle/input/animal-
kingdom/raw-img/Hen'.
There are 0 directories and 1446 images in '/kaggle/input/animal-
kingdom/raw-img/Elephant'.
There are 0 directories and 1820 images in '/kaggle/input/animal-
kingdom/raw-img/Sheep'.
There are 0 directories and 1862 images in '/kaggle/input/animal-
kingdom/raw-img/Squirrel'.
There are 0 directories and 1668 images in '/kaggle/input/animal-
kingdom/raw-img/Cat'.
```

Placing data into a Dataframe

```
def convert_path_to_df(dataset):
    image_dir = Path(dataset)

    # Get filepaths and labels
    filepaths = list(image_dir.glob(r'**/*.JPG')) +
list(image_dir.glob(r'**/*.jpg')) + list(image_dir.glob(r'**/*.jpeg'))
+ list(image_dir.glob(r'**/*.PNG'))

    labels = list(map(lambda x: os.path.split(os.path.split(x)[0])[1],
filepaths))

    filepaths = pd.Series(filepaths, name='Filepath').astype(str)
    labels = pd.Series(labels, name='Label')
```

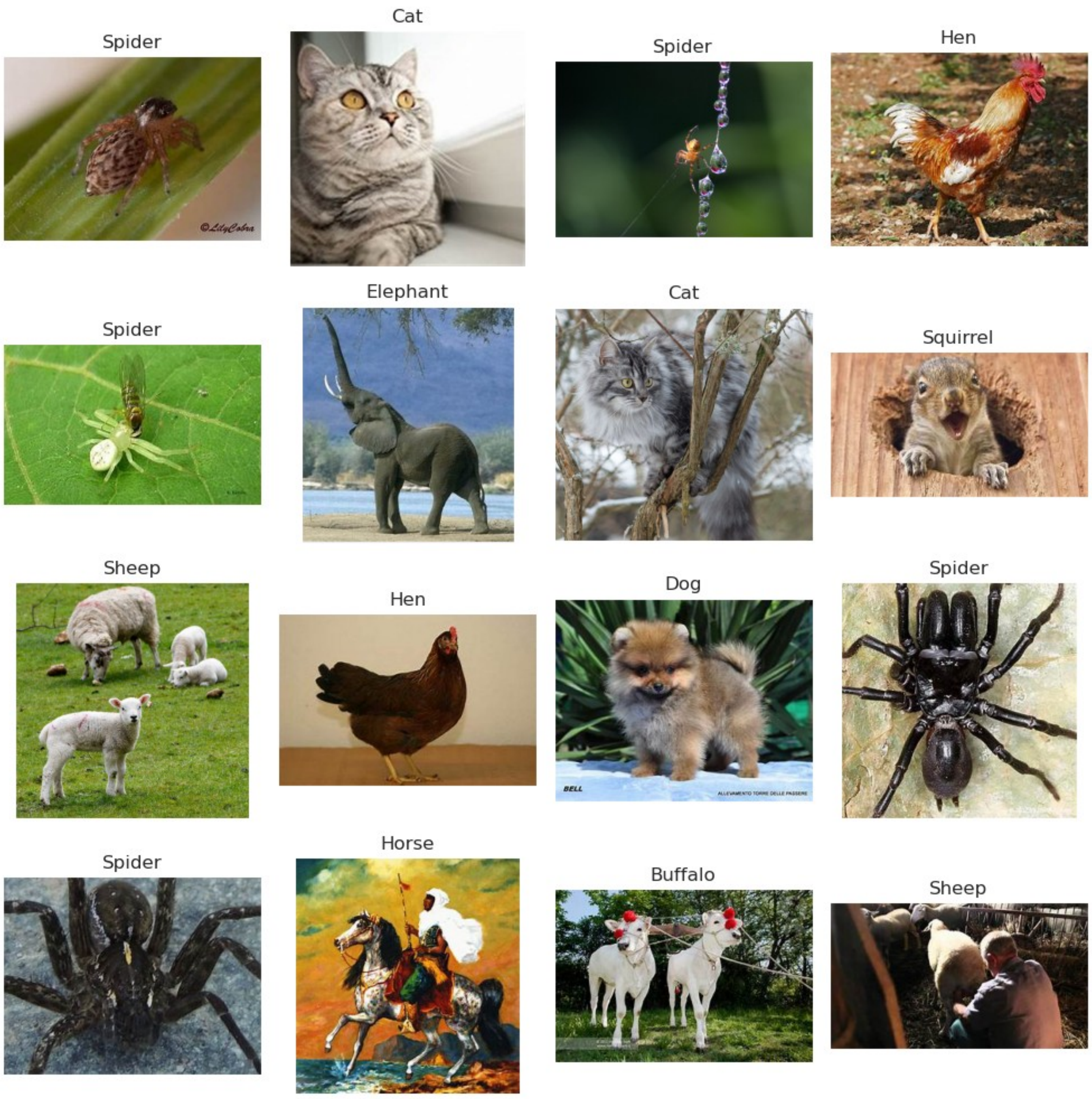
```python
    # Concatenate filepaths and labels
    image_df = pd.concat([filepaths, labels], axis=1)
    return image_df

image_df = convert_path_to_df(dataset)

# Check for corrupted images within the dataset
import PIL
from pathlib import Path
from PIL import UnidentifiedImageError

path = Path(dataset).rglob("*.jpg")
for img_p in path:
    try:
        img = PIL.Image.open(img_p)
    except PIL.UnidentifiedImageError:
            print(img_p)

# Get the value counts for each label
label_counts = image_df['Label'].value_counts()

# Create the figure and axes
fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(20, 6))

# Plot the bar chart
sns.barplot(x=label_counts.index, y=label_counts.values, alpha=0.8,
palette='pastel', ax=axes)
axes.set_title('Distribution of Labels in Image Dataset', fontsize=16)
axes.set_xlabel('Label', fontsize=14)
axes.set_ylabel('Count', fontsize=14)
axes.set_xticklabels(label_counts.index, rotation=45)

# Add a super-title to the figure
fig.suptitle('Image Dataset Label Distribution', fontsize=20)

# Adjust the spacing between the plots and the title
fig.subplots_adjust(top=0.85)

# Display the plot
plt.show()
```

Image Dataset Label Distribution



🔭Visualizing images from the dataset

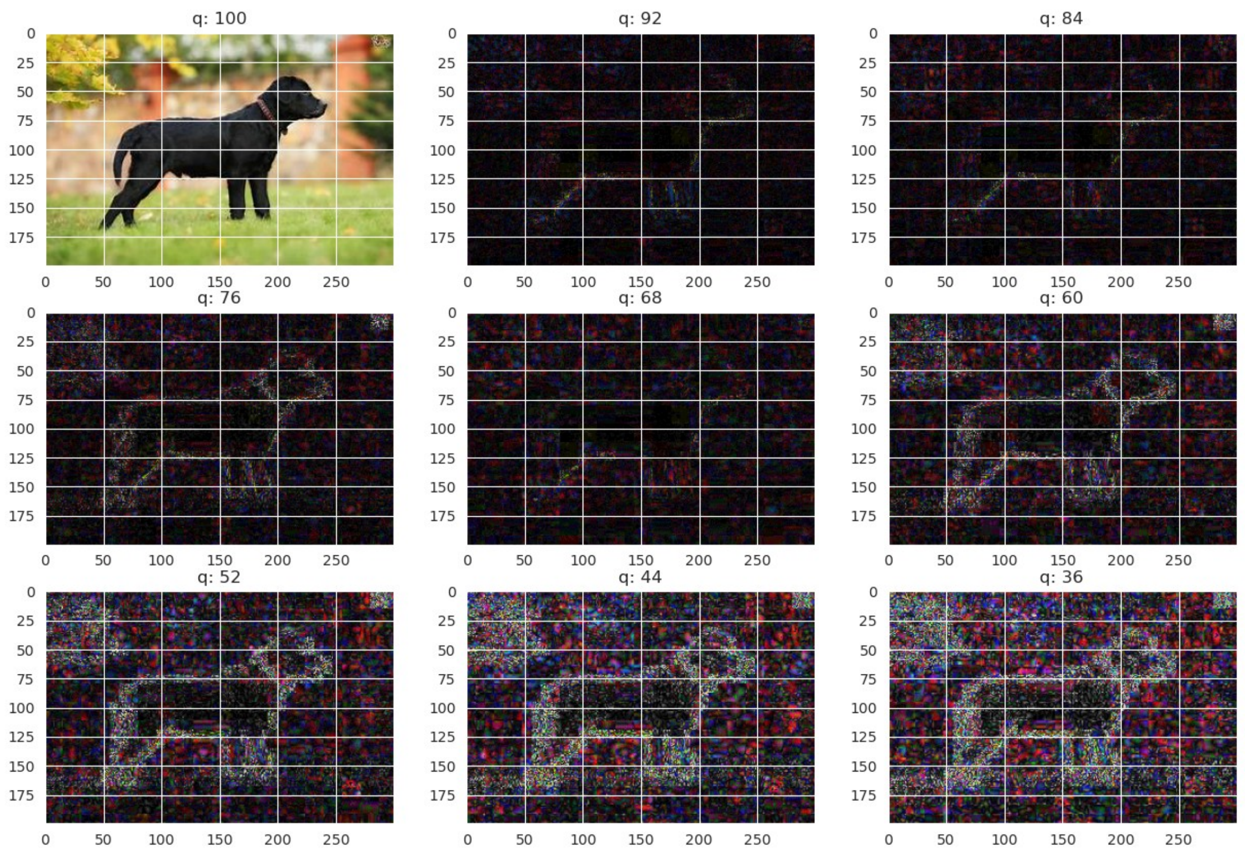
```
# Display 16 picture of the dataset with their labels
random_index = np.random.randint(0, len(image_df), 16)
fig, axes = plt.subplots(nrows=4, ncols=4, figsize=(10, 10),
                        subplot_kw={'xticks': [], 'yticks': []})

for i, ax in enumerate(axes.flat):
    ax.imshow(plt.imread(image_df.Filepath[random_index[i]]))
    ax.set_title(image_df.Label[random_index[i]])
plt.tight_layout()
plt.show()
```



Computing Error Rate Analysis

```
def compute_ela_cv(path, quality):
    temp_filename = 'temp_file_name.jpeg'
    SCALE = 15
    orig_img = cv2.imread(path)
    orig_img = cv2.cvtColor(orig_img, cv2.COLOR_BGR2RGB)

    cv2.imwrite(temp_filename, orig_img, [cv2.IMWRITE_JPEG_QUALITY, 
quality])
```

```
    # read compressed image
    compressed_img = cv2.imread(temp_filename)

    # get absolute difference between img1 and img2 and multiply by
scale
    diff = SCALE * cv2.absdiff(orig_img, compressed_img)
    return diff


def convert_to_ela_image(path, quality):
    temp_filename = 'temp_file_name.jpeg'
    ela_filename = 'temp_ela.png'
    image = Image.open(path).convert('RGB')
    image.save(temp_filename, 'JPEG', quality = quality)
    temp_image = Image.open(temp_filename)

    ela_image = ImageChops.difference(image, temp_image)

    extrema = ela_image.getextrema()
    max_diff = max([ex[1] for ex in extrema])
    if max_diff == 0:
        max_diff = 1

    scale = 255.0 / max_diff
    ela_image = ImageEnhance.Brightness(ela_image).enhance(scale)

    return ela_image


def random_sample(path, extension=None):
    if extension:
        items = Path(path).glob(f'*.{extension}')
    else:
        items = Path(path).glob(f'*')

    items = list(items)

    p = random.choice(items)
    return p.as_posix()

# View random sample from the dataset
p = random_sample('/kaggle/input/animals10/raw-img/cane')
orig = cv2.imread(p)
orig = cv2.cvtColor(orig, cv2.COLOR_BGR2RGB) / 255.0
init_val = 100
columns = 3
rows = 3

fig=plt.figure(figsize=(15, 10))
for i in range(1, columns*rows +1):
    quality=init_val - (i-1) * 8
    img = compute_ela_cv(path=p, quality=quality)
    if i == 1:
        img = orig.copy()
    ax = fig.add_subplot(rows, columns, i)
    ax.title.set_text(f'q: {quality}')
    plt.imshow(img)
plt.show()
```



Data Preprocessing

```
# Separate in train and test data
train_df, test_df = train_test_split(image_df, test_size=0.2,
shuffle=True, random_state=42)

train_generator = ImageDataGenerator(

preprocessing_function=tf.keras.applications.efficientnet.preprocess_i
nput,
    validation_split=0.2
)
```

```
test_generator = ImageDataGenerator(

preprocessing_function=tf.keras.applications.efficientnet.preprocess_i
nput,
)

# Split the data into three categories.
train_images = train_generator.flow_from_dataframe(
    dataframe=train_df,
    x_col='Filepath',
    y_col='Label',
    target_size=TARGET_SIZE,
    color_mode='rgb',
    class_mode='categorical',
    batch_size=BATCH_SIZE,
    shuffle=True,
    seed=42,
    subset='training'
)

val_images = train_generator.flow_from_dataframe(
    dataframe=train_df,
    x_col='Filepath',
    y_col='Label',
    target_size=TARGET_SIZE,
    color_mode='rgb',
    class_mode='categorical',
    batch_size=BATCH_SIZE,
    shuffle=True,
    seed=42,
    subset='validation'
)

test_images = test_generator.flow_from_dataframe(
    dataframe=test_df,
    x_col='Filepath',
    y_col='Label',
    target_size=TARGET_SIZE,
    color_mode='rgb',
    class_mode='categorical',
    batch_size=BATCH_SIZE,
    shuffle=False
)
```

```
Found 16722 validated image filenames belonging to 10 classes.
Found 4180 validated image filenames belonging to 10 classes.
Found 5226 validated image filenames belonging to 10 classes.
```

```
# Data Augmentation Step
augment = tf.keras.Sequential([
  layers.experimental.preprocessing.Resizing(224,224),
  layers.experimental.preprocessing.Rescaling(1./255),
  layers.experimental.preprocessing.RandomFlip("horizontal"),
  layers.experimental.preprocessing.RandomRotation(0.1),
  layers.experimental.preprocessing.RandomZoom(0.1),
  layers.experimental.preprocessing.RandomContrast(0.1),
])
```

🤹Training the model

```
# Load the pretained model
pretrained_model = tf.keras.applications.efficientnet.EfficientNetB7(
    input_shape=(224, 224, 3),
    include_top=False,
    weights='imagenet',
    pooling='max'
)

pretrained_model.trainable = False
```

```
Downloading data from https://storage.googleapis.com/keras-
applications/efficientnetb7_notop.h5
258076736/258076736 [==============================] - 6s 0us/step
```

```
# Create checkpoint callback
checkpoint_path = "animals_classification_model_checkpoint"
checkpoint_callback = ModelCheckpoint(checkpoint_path,
                                      save_weights_only=True,
                                      monitor="val_accuracy",
                                      save_best_only=True)

# Setup EarlyStopping callback to stop training if model's val_loss
doesn't improve for 3 epochs
early_stopping = EarlyStopping(monitor = "val_loss", # watch the val
loss metric
                               patience = 5,
                               restore_best_weights = True) # if val
loss decreases for 3 epochs in a row, stop training

reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2,
patience=3, min_lr=1e-6)
```

🚅Training the model

```
inputs = pretrained_model.input
x = augment(inputs)
```

```
x = Dense(128, activation='relu')(pretrained_model.output)
x = BatchNormalization()(x)
x = Dropout(0.45)(x)
x = Dense(256, activation='relu')(x)
x = BatchNormalization()(x)
x = Dropout(0.45)(x)


outputs = Dense(10, activation='softmax')(x)

model = Model(inputs=inputs, outputs=outputs)

model.compile(
    optimizer=Adam(0.00001),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

history = model.fit(
    train_images,
    steps_per_epoch=len(train_images),
    validation_data=val_images,
    validation_steps=len(val_images),
    epochs=100,
    callbacks=[
        early_stopping,
        create_tensorboard_callback("training_logs",
                                    "animals_classification"),
        checkpoint_callback,
        reduce_lr
    ]
)
```

```
Saving TensorBoard log files to:
training_logs/animals_classification/20240415-080336
Epoch 1/100

2024-04-15 08:03:58.307469: E
tensorflow/core/grappler/optimizers/meta_optimizer.cc:954] layout
failed: INVALID_ARGUMENT: Size of values 0 does not match size of
permutation 4 @ fanin shape inmodel/block1b_drop/dropout/SelectV2-2-
TransposeNHWCToNCHW-LayoutOptimizer

523/523 [==============================] - 185s 297ms/step - loss:
2.4286 - accuracy: 0.2920 - val_loss: 0.9384 - val_accuracy: 0.7486 -
lr: 1.0000e-05
Epoch 2/100
523/523 [==============================] - 146s 279ms/step - loss:
1.3970 - accuracy: 0.5595 - val_loss: 0.5347 - val_accuracy: 0.8787 -
lr: 1.0000e-05
```

```
Epoch 3/100
523/523 [==============================] - 146s 280ms/step - loss:
0.9723 - accuracy: 0.6942 - val_loss: 0.3605 - val_accuracy: 0.9203 -
lr: 1.0000e-05
Epoch 4/100
523/523 [==============================] - 146s 279ms/step - loss:
0.7488 - accuracy: 0.7737 - val_loss: 0.2827 - val_accuracy: 0.9371 -
lr: 1.0000e-05
Epoch 5/100
523/523 [==============================] - 146s 279ms/step - loss:
0.6082 - accuracy: 0.8165 - val_loss: 0.2373 - val_accuracy: 0.9459 -
lr: 1.0000e-05
Epoch 6/100
523/523 [==============================] - 146s 279ms/step - loss:
0.5078 - accuracy: 0.8532 - val_loss: 0.2030 - val_accuracy: 0.9519 -
lr: 1.0000e-05
Epoch 7/100
523/523 [==============================] - 146s 278ms/step - loss:
0.4542 - accuracy: 0.8717 - val_loss: 0.1862 - val_accuracy: 0.9550 -
lr: 1.0000e-05
Epoch 8/100
523/523 [==============================] - 143s 274ms/step - loss:
0.3980 - accuracy: 0.8867 - val_loss: 0.1704 - val_accuracy: 0.9545 -
lr: 1.0000e-05
Epoch 9/100
523/523 [==============================] - 146s 279ms/step - loss:
0.3683 - accuracy: 0.8970 - val_loss: 0.1575 - val_accuracy: 0.9577 -
lr: 1.0000e-05
Epoch 10/100
523/523 [==============================] - 146s 279ms/step - loss:
0.3366 - accuracy: 0.9059 - val_loss: 0.1482 - val_accuracy: 0.9593 -
lr: 1.0000e-05
Epoch 11/100
523/523 [==============================] - 146s 279ms/step - loss:
0.3112 - accuracy: 0.9148 - val_loss: 0.1399 - val_accuracy: 0.9600 -
lr: 1.0000e-05
Epoch 12/100
523/523 [==============================] - 146s 279ms/step - loss:
0.2948 - accuracy: 0.9199 - val_loss: 0.1337 - val_accuracy: 0.9627 -
lr: 1.0000e-05
Epoch 13/100
523/523 [==============================] - 146s 279ms/step - loss:
0.2707 - accuracy: 0.9285 - val_loss: 0.1294 - val_accuracy: 0.9632 -
lr: 1.0000e-05
Epoch 14/100
523/523 [==============================] - 146s 279ms/step - loss:
0.2597 - accuracy: 0.9289 - val_loss: 0.1255 - val_accuracy: 0.9648 -
lr: 1.0000e-05
Epoch 15/100
```

```
523/523 [==============================] - 146s 278ms/step - loss:
0.2428 - accuracy: 0.9344 - val_loss: 0.1222 - val_accuracy: 0.9656 -
lr: 1.0000e-05
Epoch 16/100
523/523 [==============================] - 146s 280ms/step - loss:
0.2370 - accuracy: 0.9342 - val_loss: 0.1213 - val_accuracy: 0.9665 -
lr: 1.0000e-05
Epoch 17/100
523/523 [==============================] - 146s 279ms/step - loss:
0.2265 - accuracy: 0.9379 - val_loss: 0.1173 - val_accuracy: 0.9670 -
lr: 1.0000e-05
Epoch 18/100
523/523 [==============================] - 146s 280ms/step - loss:
0.2194 - accuracy: 0.9407 - val_loss: 0.1161 - val_accuracy: 0.9677 -
lr: 1.0000e-05
Epoch 19/100
523/523 [==============================] - 147s 280ms/step - loss:
0.2035 - accuracy: 0.9444 - val_loss: 0.1135 - val_accuracy: 0.9684 -
lr: 1.0000e-05
Epoch 20/100
523/523 [==============================] - 144s 276ms/step - loss:
0.2052 - accuracy: 0.9471 - val_loss: 0.1128 - val_accuracy: 0.9682 -
lr: 1.0000e-05
Epoch 21/100
523/523 [==============================] - 146s 279ms/step - loss:
0.1950 - accuracy: 0.9468 - val_loss: 0.1112 - val_accuracy: 0.9701 -
lr: 1.0000e-05
Epoch 22/100
523/523 [==============================] - 144s 275ms/step - loss:
0.1866 - accuracy: 0.9495 - val_loss: 0.1102 - val_accuracy: 0.9696 -
lr: 1.0000e-05
Epoch 23/100
523/523 [==============================] - 144s 275ms/step - loss:
0.1898 - accuracy: 0.9489 - val_loss: 0.1083 - val_accuracy: 0.9694 -
lr: 1.0000e-05
Epoch 24/100
523/523 [==============================] - 144s 274ms/step - loss:
0.1903 - accuracy: 0.9453 - val_loss: 0.1091 - val_accuracy: 0.9699 -
lr: 1.0000e-05
Epoch 25/100
523/523 [==============================] - 144s 275ms/step - loss:
0.1761 - accuracy: 0.9538 - val_loss: 0.1073 - val_accuracy: 0.9694 -
lr: 1.0000e-05
Epoch 26/100
523/523 [==============================] - 144s 275ms/step - loss:
0.1743 - accuracy: 0.9518 - val_loss: 0.1067 - val_accuracy: 0.9699 -
lr: 1.0000e-05
Epoch 27/100
523/523 [==============================] - 143s 274ms/step - loss:
```

```
0.1723 - accuracy: 0.9535 - val_loss: 0.1071 - val_accuracy: 0.9694 -
lr: 1.0000e-05
Epoch 28/100
523/523 [==============================] - 144s 275ms/step - loss:
0.1694 - accuracy: 0.9523 - val_loss: 0.1059 - val_accuracy: 0.9696 -
lr: 1.0000e-05
Epoch 29/100
523/523 [==============================] - 145s 276ms/step - loss:
0.1639 - accuracy: 0.9539 - val_loss: 0.1052 - val_accuracy: 0.9699 -
lr: 1.0000e-05
Epoch 30/100
523/523 [==============================] - 144s 275ms/step - loss:
0.1634 - accuracy: 0.9566 - val_loss: 0.1044 - val_accuracy: 0.9701 -
lr: 1.0000e-05
Epoch 31/100
523/523 [==============================] - 144s 276ms/step - loss:
0.1592 - accuracy: 0.9563 - val_loss: 0.1038 - val_accuracy: 0.9699 -
lr: 1.0000e-05
Epoch 32/100
523/523 [==============================] - 145s 278ms/step - loss:
0.1570 - accuracy: 0.9569 - val_loss: 0.1029 - val_accuracy: 0.9711 -
lr: 1.0000e-05
Epoch 33/100
523/523 [==============================] - 144s 274ms/step - loss:
0.1454 - accuracy: 0.9593 - val_loss: 0.1025 - val_accuracy: 0.9711 -
lr: 1.0000e-05
Epoch 34/100
523/523 [==============================] - 146s 279ms/step - loss:
0.1490 - accuracy: 0.9588 - val_loss: 0.1022 - val_accuracy: 0.9713 -
lr: 1.0000e-05
Epoch 35/100
523/523 [==============================] - 146s 279ms/step - loss:
0.1472 - accuracy: 0.9597 - val_loss: 0.1020 - val_accuracy: 0.9718 -
lr: 1.0000e-05
Epoch 36/100
523/523 [==============================] - 144s 275ms/step - loss:
0.1510 - accuracy: 0.9579 - val_loss: 0.0996 - val_accuracy: 0.9703 -
lr: 1.0000e-05
Epoch 37/100
523/523 [==============================] - 146s 279ms/step - loss:
0.1404 - accuracy: 0.9611 - val_loss: 0.0994 - val_accuracy: 0.9720 -
lr: 1.0000e-05
Epoch 38/100
523/523 [==============================] - 144s 275ms/step - loss:
0.1457 - accuracy: 0.9590 - val_loss: 0.1003 - val_accuracy: 0.9720 -
lr: 1.0000e-05
Epoch 39/100
523/523 [==============================] - 144s 275ms/step - loss:
0.1369 - accuracy: 0.9618 - val_loss: 0.0997 - val_accuracy: 0.9718 -
```

```
lr: 1.0000e-05
Epoch 40/100
523/523 [==============================] - 144s 275ms/step - loss: 
0.1410 - accuracy: 0.9609 - val_loss: 0.0999 - val_accuracy: 0.9708 - 
lr: 1.0000e-05
Epoch 41/100
523/523 [==============================] - 144s 274ms/step - loss: 
0.1385 - accuracy: 0.9626 - val_loss: 0.0995 - val_accuracy: 0.9720 - 
lr: 2.0000e-06
Epoch 42/100
523/523 [==============================] - 144s 275ms/step - loss: 
0.1335 - accuracy: 0.9623 - val_loss: 0.0995 - val_accuracy: 0.9715 - 
lr: 2.0000e-06
```

✓Model Evaluation

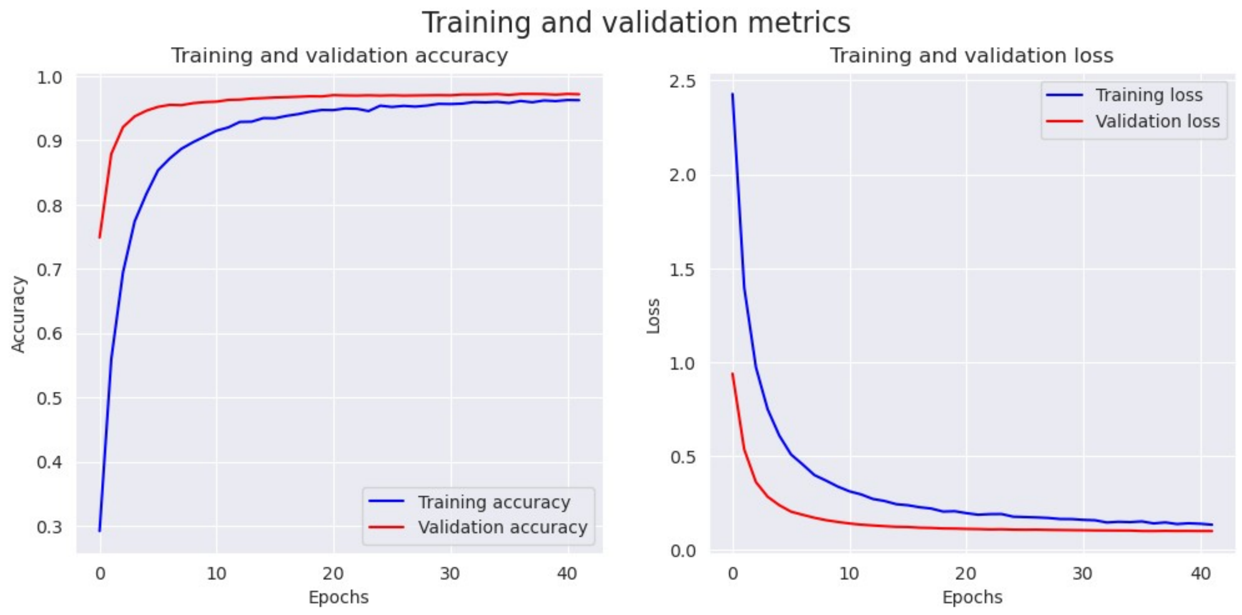
```python
results = model.evaluate(test_images, verbose=0)

print("    Test Loss: {:.5f}".format(results[0]))
print("Test Accuracy: {:.2f}%".format(results[1] * 100))
```

```
    Test Loss: 0.11451
Test Accuracy: 97.19%
```

📈Visualizing loss curves

```python
accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(accuracy))

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5))

ax1.plot(epochs, accuracy, 'b', label='Training accuracy')
ax1.plot(epochs, val_accuracy, 'r', label='Validation accuracy')
ax1.set_title('Training and validation accuracy')
ax1.set_xlabel('Epochs')
ax1.set_ylabel('Accuracy')
ax1.legend()

ax2.plot(epochs, loss, 'b', label='Training loss')
ax2.plot(epochs, val_loss, 'r', label='Validation loss')
ax2.set_title('Training and validation loss')
ax2.set_xlabel('Epochs')
```

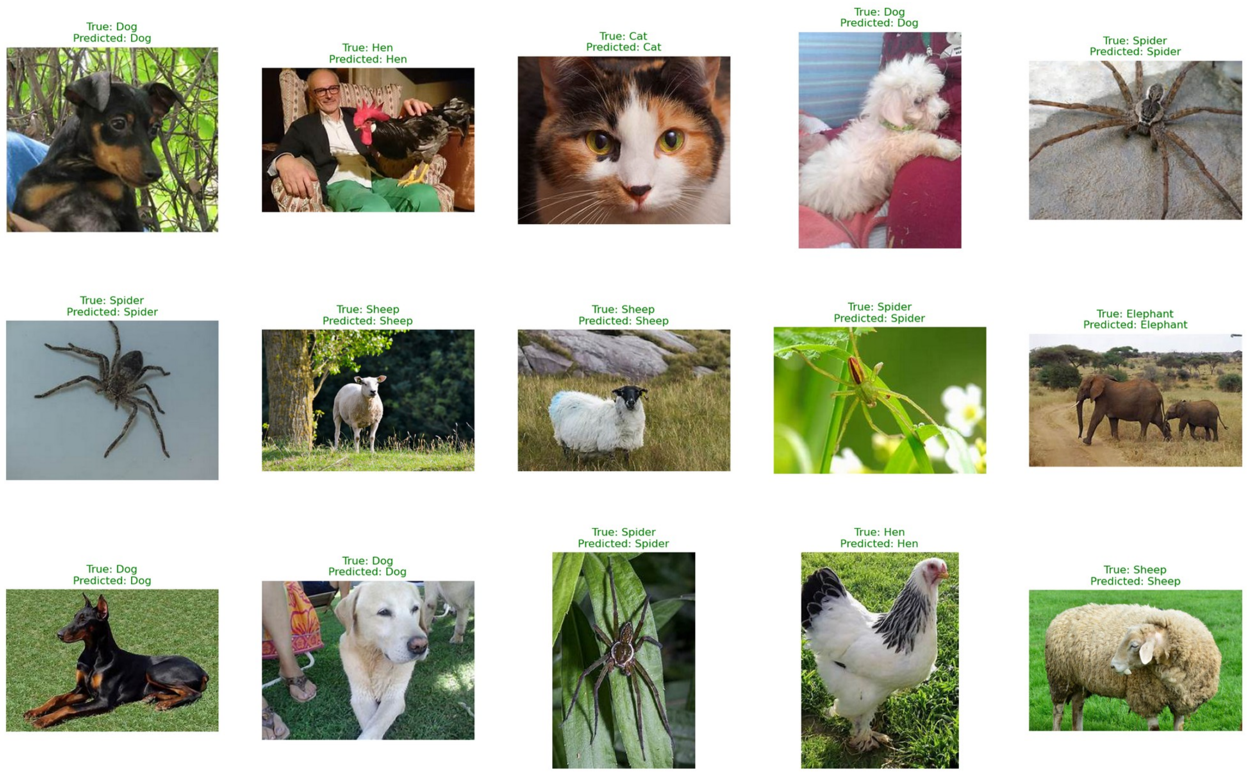
```
ax2.set_ylabel('Loss')
ax2.legend()

fig.suptitle('Training and validation metrics', fontsize=16)
plt.show()
```



Making predictions on the Test Data

```
# Predict the label of the test_images
pred = model.predict(test_images)
pred = np.argmax(pred,axis=1)

# Map the label
labels = (train_images.class_indices)
labels = dict((v,k) for k,v in labels.items())
pred = [labels[k] for k in pred]

# Display the result
print(f'The first 5 predictions: {pred[:5]}')

164/164 [==============================] - 40s 215ms/step
The first 5 predictions: ['Sheep', 'Horse', 'Dog', 'Spider',
'Squirrel']

  # Display 25 random pictures from the dataset with their labels
random_index = np.random.randint(0, len(test_df) - 1, 15)
fig, axes = plt.subplots(nrows=3, ncols=5, figsize=(25, 15),
                        subplot_kw={'xticks': [], 'yticks': []})
```

```python
for i, ax in enumerate(axes.flat):
    ax.imshow(plt.imread(test_df.Filepath.iloc[random_index[i]]))
    if test_df.Label.iloc[random_index[i]] == pred[random_index[i]]:
        color = "green"
    else:
        color = "red"
    ax.set_title(f"True: {test_df.Label.iloc[random_index[i]]}\
nPredicted: {pred[random_index[i]]}", color=color)
plt.show()
plt.tight_layout()
```



```
<Figure size 640x480 with 0 Axes>
```

📊Plotting the Classification Reports and Confusion Matrix

```python
y_test = list(test_df.Label)
print(classification_report(y_test, pred))
```

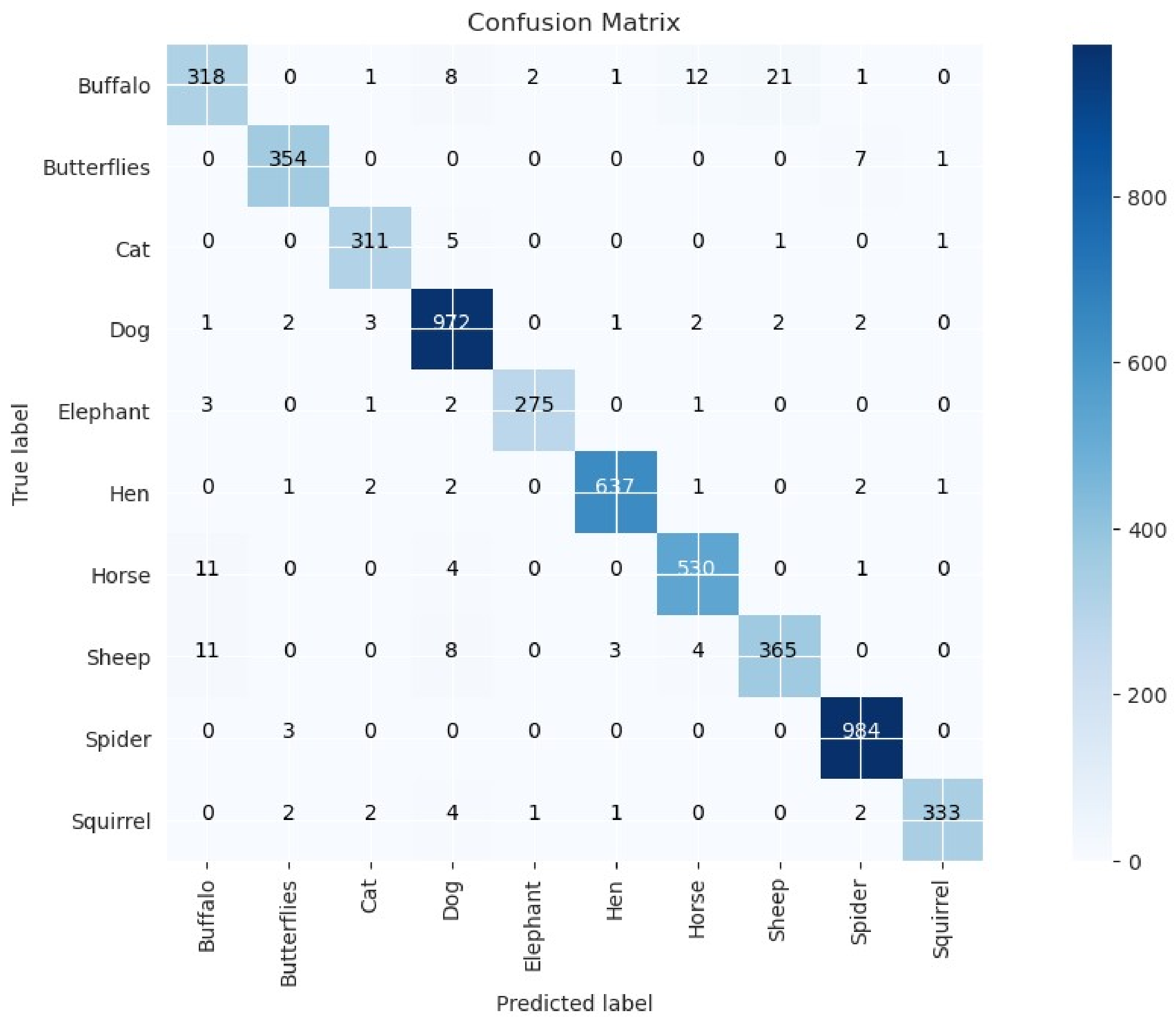
```
              precision    recall  f1-score   support

     Buffalo       0.92      0.87      0.90       364
 Butterflies       0.98      0.98      0.98       362
```

```
         Cat       0.97      0.98      0.97       318
         Dog       0.97      0.99      0.98       985
    Elephant       0.99      0.98      0.98       282
         Hen       0.99      0.99      0.99       646
       Horse       0.96      0.97      0.97       546
       Sheep       0.94      0.93      0.94       391
      Spider       0.98      1.00      0.99       987
    Squirrel       0.99      0.97      0.98       345

    accuracy                           0.97      5226
   macro avg       0.97      0.96      0.97      5226
weighted avg       0.97      0.97      0.97      5226
```

```
report = classification_report(y_test, pred, output_dict=True)
df = pd.DataFrame(report).transpose()
df
```

```
              precision    recall  f1-score      support
Buffalo        0.924419  0.873626  0.898305   364.000000
Butterflies    0.977901  0.977901  0.977901   362.000000
Cat            0.971875  0.977987  0.974922   318.000000
Dog            0.967164  0.986802  0.976884   985.000000
Elephant       0.989209  0.975177  0.982143   282.000000
Hen            0.990669  0.986068  0.988363   646.000000
Horse          0.963636  0.970696  0.967153   546.000000
Sheep          0.938303  0.933504  0.935897   391.000000
Spider         0.984985  0.996960  0.990937   987.000000
Squirrel       0.991071  0.965217  0.977974   345.000000
accuracy       0.971871  0.971871  0.971871     0.971871
macro avg      0.969923  0.964394  0.967048  5226.000000
weighted avg   0.971728  0.971871  0.971707  5226.000000
```

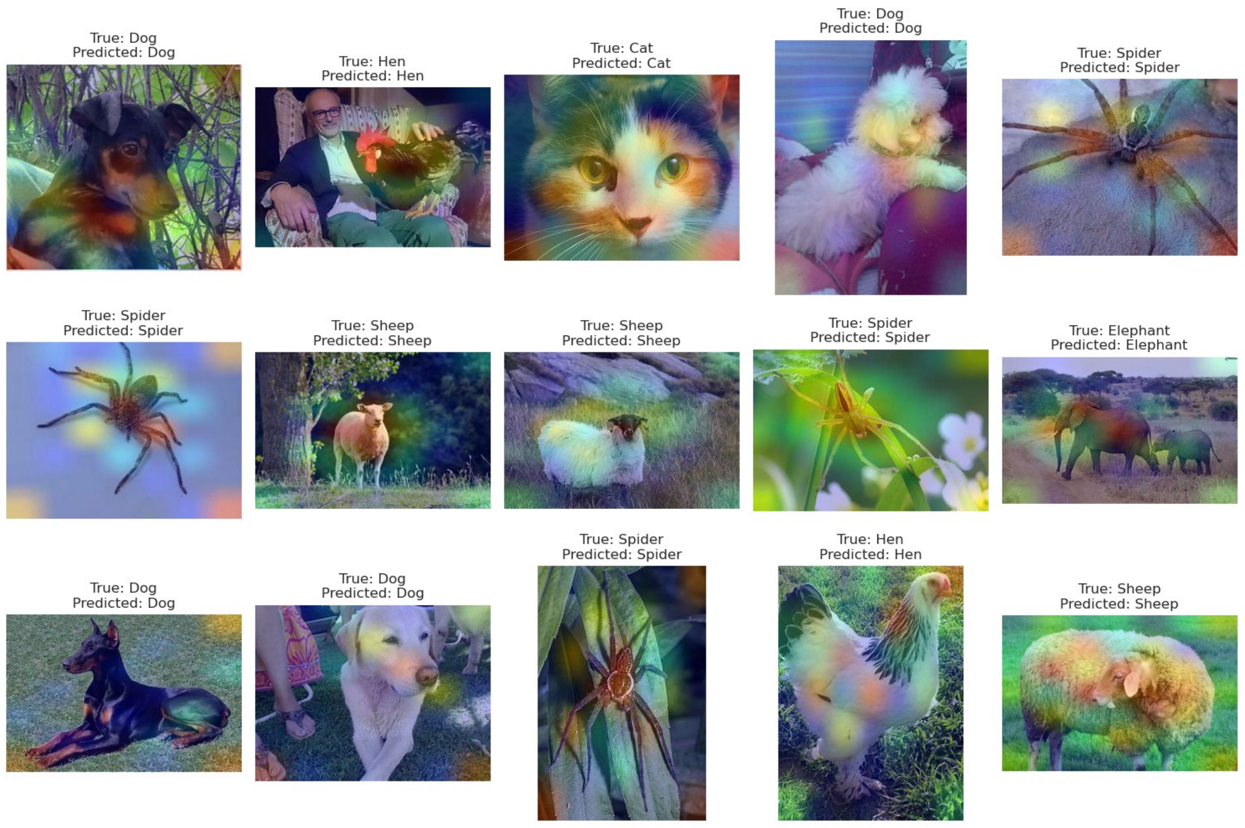
```
def make_confusion_matrix(y_true, y_pred, classes=None, figsize=(15,
7), text_size=10, norm=False, savefig=False):
    """Makes a labelled confusion matrix comparing predictions and
ground truth labels.

    If classes is passed, confusion matrix will be labelled, if not,
integer class values
  will be used.

  Args:
    y_true: Array of truth labels (must be same shape as y_pred).
    y_pred: Array of predicted labels (must be same shape as y_true).
    classes: Array of class labels (e.g. string form). If `None`,
integer labels are used.
    figsize: Size of output figure (default=(10, 10)).
    text_size: Size of output figure text (default=15).
    norm: normalize values or not (default=False).
    savefig: save confusion matrix to file (default=False).

  Returns:
    A labelled confusion matrix plot comparing y_true and y_pred.

  Example usage:
    make_confusion_matrix(y_true=test_labels, # ground truth test
labels
                          y_pred=y_preds, # predicted labels
                          classes=class_names, # array of class label
names
                          figsize=(15, 15),
                          text_size=10)
    """
  # Create the confustion matrix
    cm = confusion_matrix(y_true, y_pred)
    cm_norm = cm.astype("float") / cm.sum(axis=1)[:, np.newaxis] #
normalize it
    n_classes = cm.shape[0] # find the number of classes we're dealing
with

    # Plot the figure and make it pretty
    fig, ax = plt.subplots(figsize=figsize)
    cax = ax.matshow(cm, cmap=plt.cm.Blues) # colors will represent
how 'correct' a class is, darker == better
    fig.colorbar(cax)

    # Are there a list of classes?
    if classes:
        labels = classes
    else:
        labels = np.arange(cm.shape[0])

    # Label the axes
    ax.set(title="Confusion Matrix",
         xlabel="Predicted label",
         ylabel="True label",
         xticks=np.arange(n_classes), # create enough axis slots for
each class
         yticks=np.arange(n_classes),
         xticklabels=labels, # axes will labeled with class names (if
they exist) or ints
         yticklabels=labels)

    # Make x-axis labels appear on bottom
    ax.xaxis.set_label_position("bottom")
    ax.xaxis.tick_bottom()
    ### Added: Rotate xticks for readability & increase font size
(required due to such a large confusion matrix)
    plt.xticks(rotation=90, fontsize=text_size)
    plt.yticks(fontsize=text_size)

    # Set the threshold for different colors
    threshold = (cm.max() + cm.min()) / 2.

    # Plot the text on each cell
    for i, j in itertools.product(range(cm.shape[0]),
range(cm.shape[1])):
        if norm:
            plt.text(j, i, f"{cm[i, j]} ({cm_norm[i, j]*100:.1f}%)",
                horizontalalignment="center",
                color="white" if cm[i, j] > threshold else "black",
                size=text_size)
        else:
            plt.text(j, i, f"{cm[i, j]}",
              horizontalalignment="center",
              color="white" if cm[i, j] > threshold else "black",
              size=text_size)

  # Save the figure to the current working directory
    if savefig:
        fig.savefig("confusion_matrix.png")

make_confusion_matrix(y_test, pred, list(labels.values()))
```



	Buffalo	Butterflies	Cat	Dog	Elephant	Hen	Horse	Sheep	Spider	Squirrel
Buffalo	318	0	1	8	2	1	12	21	1	0
Butterflies	0	354	0	0	0	0	0	0	7	1
Cat	0	0	311	5	0	0	0	1	0	1
Dog	1	2	3	972	0	1	2	2	2	0
Elephant	3	0	1	2	275	0	1	0	0	0
Hen	0	1	2	2	0	637	1	0	2	1
Horse	11	0	0	4	0	0	530	0	1	0
Sheep	11	0	0	8	0	3	4	365	0	0
Spider	0	3	0	0	0	0	0	0	984	0
Squirrel	0	2	2	4	1	1	0	0	2	333

☼Grad-Cam Visualization

```
def get_img_array(img_path, size):
    img = tf.keras.preprocessing.image.load_img(img_path, 
target_size=size)
    array = tf.keras.preprocessing.image.img_to_array(img)
    # We add a dimension to transform our array into a "batch"
    # of size "size"
    array = np.expand_dims(array, axis=0)
    return array

def make_gradcam_heatmap(img_array, model, last_conv_layer_name, 
pred_index=None):
    # First, we create a model that maps the input image to the 
activations
    # of the last conv layer as well as the output predictions
    grad_model = tf.keras.models.Model(
        [model.inputs], [model.get_layer(last_conv_layer_name).output,
model.output]
    )

    # Then, we compute the gradient of the top predicted class for our
input image
    # with respect to the activations of the last conv layer
    with tf.GradientTape() as tape:
        last_conv_layer_output, preds = grad_model(img_array)
        if pred_index is None:
            pred_index = tf.argmax(preds[0])
        class_channel = preds[:, pred_index]
    # This is the gradient of the output neuron (top predicted or
chosen)
    # with regard to the output feature map of the last conv layer
    grads = tape.gradient(class_channel, last_conv_layer_output)

    # This is a vector where each entry is the mean intensity of the
gradient
    # over a specific feature map channel
    pooled_grads = tf.reduce_mean(grads, axis=(0, 1, 2))

    # We multiply each channel in the feature map array
    # by "how important this channel is" with regard to the top
predicted class
    # then sum all the channels to obtain the heatmap class activation
    last_conv_layer_output = last_conv_layer_output[0]
    heatmap = last_conv_layer_output @ pooled_grads[..., tf.newaxis]
    heatmap = tf.squeeze(heatmap)

    # For visualization purpose, we will also normalize the heatmap
between 0 & 1
    heatmap = tf.maximum(heatmap, 0) / tf.math.reduce_max(heatmap)
    return heatmap.numpy()
def save_and_display_gradcam(img_path, heatmap, cam_path="cam.jpg",
alpha=0.4):
    # Load the original image
    img = tf.keras.preprocessing.image.load_img(img_path)
    img = tf.keras.preprocessing.image.img_to_array(img)

    # Rescale heatmap to a range 0-255
    heatmap = np.uint8(255 * heatmap)

    # Use jet colormap to colorize heatmap
    jet = cm.get_cmap("jet")

    # Use RGB values of the colormap
    jet_colors = jet(np.arange(256))[:, :3]
    jet_heatmap = jet_colors[heatmap]
    # Create an image with RGB colorized heatmap
    jet_heatmap =
tf.keras.preprocessing.image.array_to_img(jet_heatmap)
    jet_heatmap = jet_heatmap.resize((img.shape[1], img.shape[0]))
    jet_heatmap =
tf.keras.preprocessing.image.img_to_array(jet_heatmap)

    # Superimpose the heatmap on original image
    superimposed_img = jet_heatmap * alpha + img
    superimposed_img =
tf.keras.preprocessing.image.array_to_img(superimposed_img)
    # Save the superimposed image
    superimposed_img.save(cam_path)

    # Display Grad CAM
#     display(Image(cam_path))

    return cam_path


preprocess_input = tf.keras.applications.efficientnet.preprocess_input
decode_predictions =
tf.keras.applications.efficientnet.decode_predictions

last_conv_layer_name = "top_conv"
img_size = (224,224, 3)

# Remove last layer's softmax
model.layers[-1].activation = None

# Display the part of the pictures used by the neural network to
classify the pictures
fig, axes = plt.subplots(nrows=3, ncols=5, figsize=(15, 10),
                        subplot_kw={'xticks': [], 'yticks': []})

for i, ax in enumerate(axes.flat):
    img_path = test_df.Filepath.iloc[random_index[i]]
    img_array = preprocess_input(get_img_array(img_path,
size=img_size))
    heatmap = make_gradcam_heatmap(img_array, model,
last_conv_layer_name)
    cam_path = save_and_display_gradcam(img_path, heatmap)
    ax.imshow(plt.imread(cam_path))
    ax.set_title(f"True: {test_df.Label.iloc[random_index[i]]}\
nPredicted: {pred[random_index[i]]}")
plt.tight_layout()
plt.show()
```



```python
# Save the entire model as a SavedModel
tf.saved_model.save(model, "animals_classification_model")
```